Les instructions de base

Pourquoi un cours d'algorithmique (1)

- Objectif : obtenir de la "machine" qu'elle effectue un travail à notre place.

- Difficultés : expliquer à la machine comment elle doit s'y prendre :
  - Mais comment le dire ?
  - Comment le lui apprendre ?
  - Comment s'assurer qu'elle fait ce travail aussi bien que nous ?

Pourquoi un cours d'algorithmique (2)

- Objectif de cet enseignement:
  - Résoudre des problèmes "comme" une machine
  - Savoir expliquer son raisonnement
  - Savoir formaliser son raisonnement
  - Concevoir et écrire des algorithmes.

C'est quoi un algorithme

- L'algorithmique vous le pratiquez tous les jours, depuis longtemps
  - Meuble en kit → (notice de montage) → cuisine équipée
  - Farine, oeufs, chocolat, etc → (recette) → Forêt noire

- Algorithme = suite de calcul pour résoudre un problème
  - Par application d'un **nombre fini** d'opérations
  - Sur un certain **nombre de données**

- Un algorithme traduit dans un langage compréhensible par l'ordinateur (ou langage de programmation) donne un **programme**, qui peut être ensuite exécuté pour effectuer un **traitement** souhaité.

Un premier algorithme

Algorithme Carré
Variables unNombre, sonCarré : entiers {déclarations : Réservation d'espace mémoire}

début

{préparation du traitement}

ecrire("Quel nombre voulez-vous élever au carré?")
lire(unNombre)

{traitement : calcul du carré}

SonCarré ← unNombre×unNombre

{présentation du résultat}

ecrire("Le carré de ", unNombre, "c'est ", sonCarré)

fin

Les trois étapes d’un algorithme

- Préparation du traitement
  - données nécessaires à la résolution du problème
- Traitement
  - résolution pas à pas, après décomposition en sous-problèmes si nécessaire
  -
- Edition des résultats
  - impression à l’écran, dans un fichier, etc.

Variables et expressions

- Une **variable est une zone de la mémoire** dans laquelle on peut stocker des données.

Variable <liste d'identificateurs> : **type**

- La taille de cette zone dépend du **type de la donnée** (et pas de la valeur qui est stockée)

- Exemples :

Variables Age, note : entiers

nom, prénom : chaînes de caractères

Variables et expressions

- Expression : suite bien formée d'opérateurs portant sur des variables ou des valeurs et qui a une valeur.
- Divers opérateurs pour manipulater des données :
  - Opérateurs arithmétiques (+ - * / mod div)
  - Opérateurs de comparaison (< >, =, <>, >=, >=)
  - Opérateurs booléens (And, Or, Not)
- Une expression est d'un certain type.

  si X une variable de type entier a la valeur 4, alors :

  1. L'expression 4 a la valeur 4
  2. L'expression X a la valeur 4
  3. L'expression X*X a la valeur 16
  4. L'expression (X+(X*2)) <= 9 a la valeur FAUX

Représentation des variables en mémoire

- Toute variable manipulée dans un programme est stockée quelque part en mémoire centrale.
- Cette mémoire est constituée d'octets qui sont identifiés de manière unique par un numéro qu'on appelle adresse.
- Pour retrouver une variable, il suffit donc de connaître l'adresse de l'octet où elle est stockée (ou, s'il s'agit d'une variable qui recouvre plusieurs octets contigus, l'adresse du premier de ces octets).
- Pour des raisons évidentes de lisibilité, on désigne souvent les variables par des identificateurs, et non par leur adresse.
  - C'est le compilateur qui fait alors le lien entre l'identificateur d'une variable et son adresse en mémoire.

Adresse et variable

- Toute variable est donc caractérisée par : Son adresse, c'est à dire l'adresse-mémoire à partir de laquelle l'objet est stocké ;
  - sa valeur, c'est-à-dire ce qui est stocké à cette adresse.
  - Son adresse, c'est à dire l'adresse-mémoire à partir de laquelle la variable est stockée ;
  - sa valeur, c'est-à-dire ce qui est stocké à cette adresse.

- Dans l'exemple

```
Variables age, note : entiers ;
age = 13;
note = 15;
```

variable	adresse	valeur
`age`	6000	`13`
`note`	6004	`15`

  - La variable `age` est placée à l'adresse 6000 en mémoire, et la variable `note` à l'adresse 6004
  - Les variables `age` et `note` étant de type `entier`, elles sont stockées sur 4 octets. Ainsi la valeur de age est stockée sur les octets d'adresse 6000 à 6003.
  - L'adresse d'une variable étant un numéro d'octet en mémoire, il s'agit d'un entier quelque soit le type de variable considérée. Le format interne de cet entier (16 bits, 32 bits ou 64 bits) dépend des architectures.

Types utilisés

Types	**Opérations sur les types**
• Entier:	+, *, - , /(division entière : 7/3 → 2), mod(modolu), inf, sup, ≤, ≥, ==, ≠
• Réel:	idem sauf modulo
• Booléen:	VRAI FAUX, v (OU) ∧ (ET), NOT
• Caractère:	inf, sup, ≤, ≥, ==, ≠
• Chaine:	idem, + à venir

Le Type BOOLEEN

- Deux constantes booléennes : VRAI , FAUX

- Des variables de type booléens :

  Variables ok, stop: booléen

  ok ← (rep= ‘O’ou rep= ‘o’)

  stop ← (val ≤ 0 ou val ≥ 9)

- Dans les conditionnelles et itératives :

  Tant que ok faire…

  Si non(stop) alors...

Éléments sur les opérations logiques

- Formule : expression logique composée de variables propositionnelles et de connecteurs logiques

- Variable propositionnelle : proposition indécomposable

- Connecteurs logiques:
  - négation non,¬
  - conjonction et, ∧
  - disjonction ou, ∨
  - Ou exclusif, XOR (e**X**clusive **OR**)
- Exemple : soit A et B variables propositionnelles

  (A ∧¬B) ∨ (¬A ∨B)

Table de vérité

Représentation des valeurs de vérité
associées à une expression logique

Négation

A	¬A
V	F
F	V

Conjonction

A	B	A ∧ B
V	V	V
V	F	F
F	V	F
F	F	F

Disjonction

A	B	A ∨ B
V	V	V
V	F	V
F	V	V
F	F	F

XOR

A	B	A ⊕ B
V	V	F
V	F	V
F	V	V
F	F	F

$\neg (p \wedge q) \leftrightarrow (\neg p) \vee (\neg q)$

$\neg (p \vee q) \leftrightarrow (\neg p) \wedge (\neg q)$

Instruction d'affectation et expression

- 1 instruction = spécification d'1 ou plusieurs actions portant sur les variables.

- Instruction d'affectation : consiste à doter une variable d'une valeur de son domaine.

Variable ← <expression> ou <identificateur> ou <constante>

- Expression = suite bien formée d'opérations portant sur des variables ou des valeurs.
- Exemple : si X une variable de type entier a la valeur 4, alors
  a) L'expression X a la valeur 4
  b) L'expression X*X a la valeur 16
  c) L'expression (X+(X*2)) ≤ 9 a la valeur FAUX
  L'expression c) est une dite **expression logique**

Déclaration d'une constante

- Instruction permettant de réserver de l'espace mémoire pour stocker des données dont la valeur est fixée pour tout l'algorithme

constante (<identificateur> : type) ← <expression>

- Exemples :

constantes TVA : réel ← 19.6
MAX: entier ← 100
SeuilMAX : entier ← MAX × 0.2

Instruction de lecture/écriture

- Pour Lire 1 valeur sur un périphérique d'Entrée

Lire(variable)

- Pour écrire 1 valeur sur un périphérique de sortie

Ecrire(variable)

- Exemples :

L'instruction conditionnelle

```
Si  <condition>
    alors <instruction1>
    [sinon <instruction2> ]
fsi
```

- Si la condition prend la valeur vrai, le premier bloc d'instructions est exécuté; si elle prend la valeur faux, le second bloc est exécuté (s'il est présent, sinon, rien)

Exemple

Condition simple :

Afficher "recu" si note >= 10,

Afficher "insiffisant" si note < 10

```
Si  (note >= 10)
    alors Ecrire (“reçu”)
    sinon Ecrire(“insuff”)
fsi
```

Condition imbriquée : Idem que l'exemple précédent mais avec mention

```
Si  (note > 12)
   alors Ecrire (“reçu avec mention Bien”)
    sinon Si (note >= 10)
          alors Ecrire(“reçu avec mention passable”)
           Sinon Ecrire(“insuffisant”)
           Fsi
Fsi
```

Ambiguité du sinon

```
Si  (n >= 12)
    alors Si (n > 16)
        alors Ecrire (“TB”)
    sinon Ecrire(“B”)
```

Il y a ambiguité dans le sinon

Exemple de réécriture d'un si imbriqué :

```
Si  (cd1)
   alors Si (cd2)
        alors instr1
        sinon instr2
        Fsi
   Fsi
```

```
Si  (cd1 et cd2)
   alors instr1
sinon (// cd1 = F ou cd2 = F) ou (cd1 = F et cd2 = F)
        Si (cd1 et non cd2)
            alors instr2
        Fsi
Fsi
```

L'instruction selon

```
Selon <identificateur> :
        val1: instruction1
        val2: instruction2
        ...
        [defaut: ... ]
FinSelon
```

- A utiliser s'il y a plus de deux choix possibles